

2



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/706,515	11/12/2003	Fei Luo	BEAS-1339US2	7689
23910	7590	03/08/2006	EXAMINER	
FLIESLER MEYER, LLP FOUR EMBARCADERO CENTER SUITE 400 SAN FRANCISCO, CA 94111			ZHEN, LI B	
			ART UNIT	PAPER NUMBER
			2194	

DATE MAILED: 03/08/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

**Office Action Summary**

Application No.

10/706,515

Applicant(s)

LUO ET AL.

Examiner

Li B. Zhen

Art Unit

2194

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --  
Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

- 1) ☒ Responsive to communication(s) filed on 12 December 2005.
- 2a) ☒ This action is **FINAL**. 2b) ☐ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

- 4) ☒ Claim(s) 1-10,13,16-18 and 21-25 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-10,13,16-18 and 21-25 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

**Application Papers**

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on \_\_\_\_\_ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.


**Priority under 35 U.S.C. § 119**

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- \* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_.

- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_.
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: \_\_\_\_\_.

  
WILLIAM THOMSON  
SUPERVISORY PATENT EXAMINER

**DETAILED ACTION**

1. Claims 1 – 10, 13, 16 – 18, and 21 – 25 are pending in the application.

***Response to Arguments***

2. Applicant's arguments with respect to the claims have been considered but are moot in view of the new ground(s) of rejection.

***Claim Objections***

3. Claim 13 is objected to because of the following informalities: claim 13 depends from cancelled claim 12. Appropriate correction is required.

***Claim Rejections - 35 USC § 103***

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

5. **Claims 1 – 10, 13, 16 – 18, and 21 – 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,877,163 to Jones et al. [hereinafter referred to as Jones, cited in the previous office action] in view of U.S. Patent Application Publication No. 2004/0143835 to Dattke et al. [hereinafter referred to as Dattke].**

6. As to claim 1, Jones teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for dynamically generating a wrapper object [a proxy class that is dynamically generated at runtime; col. 3, lines 5 – 15], which instructions, when executed by one or more processors [col. 4, lines 36 – 52]; cause the one or more processors to carry out the steps of:

receiving a class and superclass [client invokes, for example, a Java Class Library (described below) call using a name or other identifier to obtain a class object representative of each desired interface; col. 3, line 63 – col. 4, line 17];

performing reflection on the class to obtain specific extension methods defined within the class [col. 8, lines 5 – 67] the reflection including retrieving meta information for allowing a server to generate a wrapper class that matches the vendor class [determines the interfaces that will be supported by the proxy class 202 (step 302), Fig. 3, col. 5, lines 50 – 63; Examiner notes that Reflection is an operational feature of Java that allows meta information to be retrieved from code (p. 10, paragraph 0024 of applicant's specification). Since Jones also discloses Java Reflection (col. 8, lines 5 – 67), Jones would also retrieve meta information from the code.];

generating a wrapper class as a subclass of the superclass [generating a proxy class 202 and a proxy class instance 204; col. 5, lines 50 – 62];

generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [client instantiates the proxy class by providing it an invocation handler; col. 3, line 57 – col. 4, line 17]; and

providing the wrapper object to an application program, thereby providing the application program with access [proxy class provides uniform access to the methods of the multiple types of interfaces implemented by the proxy class without regard to the type; col. 3, lines 29 – 43] to specific extension methods [Once a caller has access to the proxy class instance 204, the caller may wish to invoke a method (step 502); col. 6, lines 43 – 63]. Although Jones teaches the invention substantially as claimed, Jones does not specifically teach a wrapper class comprising at least one of vendor specific extension methods from the vendor class.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3,

Art Unit: 2194

paragraph 0031], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

7. It would have been obvious to a person of ordinary skill in the art at the time of the invention to apply the teaching of a wrapper class comprising at least one of vendor specific extension methods from the vendor class as taught by Dattke to the invention of Jones because this allows customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke] and a dynamic proxy class can be used to create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke].

8. As to claim 2, Jones teaches the wrapper object is dynamically generated at runtime [col. 2, lines 18 – 33].

9. As to claim 3, Jones teaches the superclass is one of a pre-existing JDBC, JMS, or connector class [col. 4, lines 36 – 52].

10. As to claim 4, Jones, teaches the superclass includes logic to handle server side tasks [dynamic proxy class can be created using interface specifications received from a remote server at runtime; col. 4, lines 21 – 35].

11. As to claim 5, Jones teaches the wrapper class is generated in bytecode [col. 4, lines 52 – 67].

Art Unit: 2194

12. As to claim 6, Jones teaches bytecode is generated for vendor methods not implemented in the superclass [col. 3, lines 29 – 43].

13. As to claim 7, Jones teaches the bytecode is generated using hot code generation [col. 2, lines 18 – 33].

14. As to claim 8, Jones as modified teaches providing the wrapper object to an application program, enables the application program to access standard features defined by the superclass [col. 3, lines 29 – 43 of Jones] and non-standard vendor extensions defined by the vendor defined class [p. 3, paragraph 0035 of Dattke].

15. As to claim 9, Jones teaches the standard Java Platform and distributed computing [col. 4, lines 52 – 67] but does not disclose J2EE. However, it would have been obvious to a person of ordinary skilled in the art at the time of the invention that Jones would use J2EE to take advantage of new features provided by J2EE.

16. As to claim 10, Jones as modified teaches a machine-readable medium carrying one or more sequences of instructions for processing an invocation at a dynamically generated wrapper, which instructions, when executed by one or more processors, cause the one or more processor to carry out the steps of:

receiving, from an application program, an invocation [a client wishes to make a method invocation for a method of an interface implemented by the proxy class; col. 3, lines 29 – 42 of Jones] directed to a wrapped vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035 of Dattke];

initiating pre-processing by calling a pre-invocation handler [extension factory] configured to execute server-side code [extension factory checks the entries in the

Art Unit: 2194

central extension registry to determine if there are any existing implementations for the given extension object; paragraph 0035 of Dattke];

calling the wrapped vendor object [col. 3, lines 29 – 43 of Jones];

receiving a result from the wrapped vendor object [invocation handler returns the result to the proxy class instance and then back to the client that made the original method invocation request; col. 3, lines 29 – 43 of Jones];

initiating post-processing by calling a post-invocation handler configured to execute post processing server-side tasks [result handler 310 is called after all the implementations of the extension object have been called. The result handler 310 evaluates the results from the multiple implementations of the application extension and determines the results that are returned to the standard application; p. 3, paragraph 0034 of Dattke]; and

providing the result to the application [invocation handler 122 then returns the result to the proxy class instance 204 (step 508) which then returns the result back to the caller (510); col. 6, lines 57 – 63 of Jones], thereby enabling the application program to access vendor specific extension methods of the wrapped vendor object [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031 of Dattke].

17. As to claim 13, Jones teaches the server-side code executed by the pre-invocation handler includes global transaction processing code [col. 3, line 57 – col. 4, line 17].

18. As to claim 16, Jones teaches the post-processing server-side tasks include global transaction management [col. 3, line 57 – col. 4, line 17].

19. As to claim 17, Jones teaches wherein providing the wrapper object to an application program enables the application to access wrapped vendor objects without requiring a relinking of the application and a vendor software package [client does not need to rebuild the proxy class; col. 3, lines 15 – 29].

20. As to claim 18, Jones teaches wherein calling the wrapped vendor object enables the wrapped vendor object to be processed by the application without requiring a relinking of the application and a vendor software package [client does not need to rebuild the proxy class; col. 3, lines 15 – 29].

21. As to claim 21, Jones as modified teaches a machine-readable medium carrying one or more sequences of instructions for enabling an application program to interface [a proxy class that is dynamically generated at runtime; col. 3, lines 5 – 15 of Jones] with a vendor application, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

- receiving a vendor provided class [standard application; p. 4, paragraph 0038 of Dattke] used to interface with third party software [a client wishes to make a method invocation for a method of an interface implemented by the proxy class; col. 3, lines 29 – 42 of Jones];

- preparing a wrapper object for interfacing with vendor specific extension methods [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035 of Dattke] of the vendor provided class by reflecting [col. 8, lines 5 – 67 of Jones] the vendor provided class and a superclass to form a wrapper class from which the wrapper object is instantiated [col. 3, line 63 – col. 4, line 17 of Jones]; and

- providing the wrapper object to the application program [proxy class provides uniform access to the methods of the multiple types of interfaces implemented by the proxy class without regard to the type; col. 3, lines 29 – 43 of Jones], thereby enabling the application program capability to access vendor specific extension methods of the vendor application using the wrapper object [col. 16, lines 21 – 40 of Jones].



22. As to claim 22, Jones as modified teaches a machine-readable medium carrying one or more sequences of instructions for processing an invocation at a dynamically generated wrapper [a proxy class that is dynamically generated at runtime; col. 3, lines 5 – 15 of Jones] enabling an application program to interface with a vendor application, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving, from an application program [a client wishes to make a method invocation for a method of an interface implemented by the proxy class; col. 3, lines 29 – 42 of Jones], an invocation call directed to a wrapped vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035 of Dattke];

calling the wrapped vendor object [col. 3, lines 29 – 43 of Jones];

receiving a result from the wrapped vendor object [invocation handler returns the result to the proxy class instance and then back to the client that made the original method invocation request; col. 3, lines 29 – 43 of Jones]; and

providing the result to the application program [invocation handler 122 then returns the result to the proxy class instance 204 (step 508) which then returns the result back to the caller (510); col. 6, lines 57 – 63 of Jones], thereby enabling the application program to access vendor specific extension methods of the wrapped vendor object [pp. 2-3, paragraph 0031 of Dattke].

23. As to claim 23, Jones teaches dynamically generating the wrapper class in byte code that perfectly matches with the vendor class [col. 4, line 52 – col. 5, line 5].

24. As to claim 24, Jones as modified teaches a machine-readable medium carrying one or more sequences of instructions for processing an invocation at a dynamically generated wrapper, which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving, from an application program, a method invocation call [a client wishes to make a method invocation for a method of an interface implemented by the proxy class; col. 3, lines 29 – 42 of Jones] directed to a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035 of Dattke];

calling a wrapper object [col. 3, lines 29 – 43 of Jones] for processing the method invocation call wherein the wrapper object has been dynamically generated from a vendor class to be associated with the vendor object [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031 of Dattke];

initiating pre-processing by the wrapper object, wherein the wrapper object calls a pre-invocation handler configured to perform server side logic [extension factory checks the entries in the central extension registry to determine if there are any existing implementations for the given extension object; paragraph 0035 of Dattke];

forwarding the method invocation call to the vendor object by the wrapper object on behalf of the application program [Once a caller has access to the proxy class instance 204, the caller may wish to invoke a method (step 502); col. 6, lines 43 – 63 of Jones];

receiving a result of the method invocation call from the vendor object by the wrapper object [invocation handler returns the result to the proxy class instance and then back to the client that made the original method invocation request; col. 3, lines 29 – 43 of Jones];

initiating post-processing by the wrapper object, wherein the wrapper object calls a post-invocation handler configured to perform server-side logic [result handler 310 is called after all the implementations of the extension object have been called. The result handler 310 evaluates the results from the multiple implementations of the application extension and determines the results that are returned to the standard application; p. 3, paragraph 0034 of Dattke]; and

providing the result to the application program [invocation handler 122 then returns the result to the proxy class instance 204 (step 508) which then returns the result back to the caller (510); col. 6, lines 57 – 63 of Jones], thereby enabling the application program to access vendor specific extension methods of the vendor object [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031 of Dattke].

25. As to claim 25, Jones as modified teaches the server-side logic includes at least one of global transaction management, pooling, caching, tracing and profiling [logger 315 can be used to log the result of calling the multiple implementations of the extension object; p. 3, paragraph 0034 of Dattke].

### ***Conclusion***

26. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire **THREE MONTHS** from the mailing date of this action. In the event a first reply is filed within **TWO MONTHS** of the mailing date of this final action and the advisory action is not mailed until after the end of the **THREE-MONTH** shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than **SIX MONTHS** from the date of this final action.

Art Unit: 2194

### CONTACT INFORMATION


27. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Li B. Zhen whose telephone number is (571) 272-3768. The examiner can normally be reached on Mon - Fri, 8:30am - 5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William Thomson can be reached on 571-272-3718. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Li B. Zhen  
Examiner  
Art Unit 2194

lbz

  
WILLIAM THOMSON  
SUPERVISORY PATENT EXAMINER